

1/5/2 (Item 2 from file: 351) [Links](#)

Fulltext available through: [Order File History](#)

Derwent WPI

(c) 2008 The Thomson Corporation. All rights reserved.

0010956888 & Drawing available

WPI Acc no: 2001-580012/200165

XPX Acc No: N2001-431819

**Hardware description verifying system for software development, detects variation in logic interpretation of program portion, by discrepancies of compiled program from behavioral synthesis of hardware description**

Patent Assignee: FURUSAWA S (FURU-I); NEC CORP (NIDE)

Inventor: FURUSAWA S

Patent Family ( 2 patents, 2 & countries )

Patent Number	Kind	Date	Application Number	Kind	Date	Update	Type
US 20010016935	A1	20010823	US 2001777543	A	20010206	200165	B
JP 2001222565	A	20010817	JP 200032355	A	20000209	200165	E

Priority Applications (no., kind, date): JP 200032355 A 20000209

Patent Details

Patent Number	Kind	Lan	Pgs	Draw	Filing Notes
US 20010016935	A1	EN	27	19	
JP 2001222565	A	JA	12		

**Alerting Abstract US A1**

NOVELTY - A storage unit (4) stores a source program for hardware description in a programming language. A processor detects a portion of source program which is different in logic interpretation, by verifying discrepancies between a compiled program and behavioral synthesis of hardware description, and notifies the detected result to output unit (7).

DESCRIPTION - INDEPENDENT CLAIMS are also included for the following:

- A. Hardware description verifying method;
- B. Hardware description verifying program

USE - For determining discrepancies in logic interpretations between hardware description of programming languages such as C, C++, Java and the hardware description language (HDL), for software development.

ADVANTAGE - Enables detection of different portions of hardware description without examining the equality between functional verification of hardware descriptions and the behavior synthesis. Thus, the designer or user can easily rewrite the detected portion in the source program without any discrepancies.

DESCRIPTION OF DRAWINGS - The figure shows the block diagram of hardware description verifying system.

4 Storage unit

7 Output unit

**Title Terms /Index Terms/Additional Words:** HARDWARE; DESCRIBE; VERIFICATION; SYSTEM; SOFTWARE; DEVELOP; DETECT; VARIATION; LOGIC; INTERPRETATION; PROGRAM; PORTION; DISCREPANCY; COMPILE; SYNTHESIS

**Class Codes**

International Patent Classification

IPC	Class Level	Scope	Position	Status	Version Date
-----	-------------	-------	----------	--------	--------------

G06F-017/50			Main		"Version 7"
G06F-0017/50	A	I		R	20060101
G06F-0017/50	C	I		R	20060101

**US Classification, Issued: 7164, 7165, 7166**

File Segment: EPI;

DWPI Class: T01

Manual Codes (EPI/S-X): T01-F05A; T01-J15; T01-J15B; T01-S01C; T01-S02

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-222565

(P2001-222565A)

(43) 公開日 平成13年8月17日 (2001.8.17)

(51) Int.Cl.<sup>7</sup>

G 0 6 F 17/50

識別記号

F I

G 0 6 F 15/60

テーマコード(参考)

6 6 4 Z 5 B 0 4 6

6 6 4 G

審査請求 有 請求項の数16 O L (全 12 頁)

(21) 出願番号 特願2000-32355(P2000-32355)

(22) 出願日 平成12年2月9日(2000.2.9)

(71) 出願人 000004237

日本電気株式会社

東京都港区芝五丁目7番1号

(72) 発明者 古澤 慎也

東京都港区芝五丁目7番1号 日本電気株式会社内

(74) 代理人 100102864

弁理士 工藤 実 (外1名)

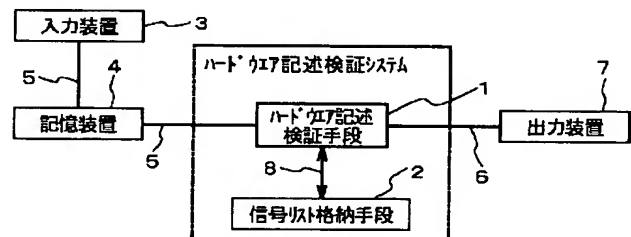
Fターム(参考) 5B046 AA08 JA03 JA05 KA01

(54) 【発明の名称】 ハードウェア記述の検証システム及びその検証方法

(57) 【要約】

【課題】 ハードウェア記述のうちプログラム言語とHDLとで解釈が分かれる部分の存在が機能検証シミュレーションを妨害することを回避する。

【解決手段】 クロック文により値が更新されるプログラム言語で記述されるハードウェア記述をコンパイルするプログラム言語と、ハードウェア記述を動作合成するHDL言語との間で論理解釈が異なる部分を検出する。解釈が異なる部分は、クロックに同期して更新が起こる被代入変数が先行する他の数式を参照する部分、クロックに同期して更新が起こる被代入変数が重ね書きされる部分、同一クロック境界内でのみ有効であり同一クロック境界を越えるとその値が無効になる非代入変数が存在する部分、非オーバーライト型であり代入された値がその同一クロック境界内の全ての参照に反映される被代入変数が存在する部分等である。解釈が異なる部分を事前に発見することにより、検証の無駄が省略される。



## 【特許請求の範囲】

【請求項 1】プログラム言語により記述したハードウェア記述をコンパイルする場合と、HDL 言語により記述したハードウェア記述を動作合成する場合とで、論理解釈が異なる部分を検出することを有するハードウェア記述の検証方法。

【請求項 2】前記解釈が異なる部分は、被代入変数への代入後に前記被代入変数を他の数式で参照することが同一のクロックタイミング内で行われる部分である請求項 1 記載のハードウェア記述の検証方法。

【請求項 3】前記プログラム言語により記述したハードウェア記述の複数文の一文を順番に入力するステップと、

前記一文が前記被代入変数への代入を行う文であると前記被代入変数を信号リストへ登録するステップと、前記一文がクロック境界であれば登録されている前記被代入変数を前記信号リストから削除するステップと、前記一文が前記信号リストに登録されている前記被代入変数を参照しているかどうかを判断するステップとを有する請求項 2 記載のハードウェア記述の検証方法。

【請求項 4】前記一文が前記信号リストに登録されている前記被代入変数を参照している場合に、前記プログラム言語によるハードウェア記述と前記 HDL 言語によるハードウェア記述との解釈が異なる動作となる警告を行うステップとを有する請求項 3 記載のハードウェア記述の検証方法。

【請求項 5】前記解釈が異なる部分は、同一クロックタイミング内において被代入変数が非オーバーライト型として重ね書きされる部分である請求項 1 記載のハードウェア記述の検証方法。

【請求項 6】前記プログラム言語により記述したハードウェア記述の複数文の一文を順番に入力するステップと、

前記一文が前記被代入変数への代入を行う文であると前記被代入変数を信号リストへ登録するステップと、前記一文がクロック境界であれば登録されている前記被代入変数を前記信号リストから削除するステップと、前記一文が前記信号リストに登録されている前記被代入変数への代入を行っているかどうかを判断するステップとを有する請求項 5 記載のハードウェア記述の検証方法。

【請求項 7】前記一文が前記信号リストに登録されている前記被代入変数への代入を行っている場合に前記プログラム言語によるハードウェア記述と前記 HDL 言語によるハードウェア記述との解釈が異なる動作となる警告を行うステップとを有する請求項 6 記載のハードウェア記述の検証方法。

【請求項 8】前記解釈が異なる部分は、同一クロックタイミング内でのみ値が有効であり、前記クロックタイミングを越えるとその値が無効になる非代入変数が存在す

る部分である請求項 1 記載のハードウェア記述の検証方法。

【請求項 9】前記解釈が異なる部分は、非オーバーライト型である前記被代入変数へ代入された値が同一クロックタイミング内の全ての前記被代入変数の参照に反映される被代入変数が存在し、且つ、代入よりも参照が先になされている部分である請求項 1 記載のハードウェア記述の検証方法。

【請求項 10】前記解釈が異なる部分は、第 1 オペランドの条件判断が第 2 オペランドの評価に依存する、両オペランドを用いる演算子が存在する部分である請求項 1 記載のハードウェア記述の検証方法。

【請求項 11】プログラム言語により記述したハードウェア記述の文が一文ずつ入力されるハードウェア記述検証装置と、

信号リスト格納装置とを含み、

前記ハードウェア記述検証装置は、前記プログラム言語によるハードウェア記述をコンパイルする場合と HDL 言語によるハードウェア記述を動作合成する場合との間で動作の解釈が異なる部分を検出し、

前記信号リスト格納装置は、前記解釈が異なる部分を格納するハードウェア記述の検証システム。

【請求項 12】前記解釈が異なる部分は、被代入変数への代入後に前記被代入変数を他の数式で参照することが同一クロックタイミング内で行われる部分である請求項 11 記載のハードウェア記述の検証システム。

【請求項 13】前記解釈が異なる部分は、同一クロックタイミング内において被代入変数が非オーバーライト型として重ね書きされる部分である請求項 11 記載のハードウェア記述の検証システム。

【請求項 14】前記解釈が異なる部分は、同一クロックタイミング内でのみ値が有効であり、前記クロックタイミングを越えるとその値が無効になる非代入変数が存在する部分である請求項 11 記載のハードウェア記述の検証システム。

【請求項 15】前記解釈が異なる部分は、非オーバーライト型である前記被代入変数へ代入された値が同一クロックタイミング内の全ての前記被代入変数の参照に反映される被代入変数が存在し、且つ、代入よりも参照が先にされている部分である請求項 11 記載のハードウェア記述の検証システム。

【請求項 16】前記解釈が異なる部分は、第 1 オペランドの条件判断が第 2 オペランドの評価に依存する、両オペランドを用いる演算子が存在する部分である請求項 11 記載のハードウェア記述の検証システム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、ハードウェア記述の検証システム及びその検証方法に関し、特に、プログラム言語でハードウェアが記述されているハードウェア

記述について、ソフトウェアプログラムとしてコンパイルした場合と、ハードウェア動作記述として合成した場合とで、実質的に相違する恐れがある相違部分を発見するハードウェア記述の検証システム及びその検証方法に関する。

#### 【0002】

【従来の技術】従来、ハードウェアの設計はハードウェア記述言語（HDL）を用いて行われ、そのハードウェアを制御するソフトウェアはプログラム言語を用いて記述されていた。ハードウェアとそのハードウェアを制御するソフトウェアプログラムとの統合検証は、協調シミュレータといわれるハードウェア記述言語とプログラム言語との混在実行が可能なシミュレータを用いて行われていた。

【0003】このようなことは、ソフトウェアプログラムのデバッグ段階においても全く同様である。機能検証ステップS101において、ソフトウェアをハード上でシミュレーション動作させることでハードウェアを機能検証してそのハードウェアをデバッグするときハードウェアを表すHDL記述とソフトウェアを表すプログラム言語とを用いて協調させると、ハードウェアシミュレーションの動作の遅さに検証時間が依存してしまう。最近では、そのような依存がなくソフトウェアもハードウェアもプログラム言語のみで記述し、高速検証を可能とするために、HDLの代わりにソフトウェアの開発に用いられるC、C++、Javaのようなプログラム言語によりハードウェア動作を記述して、ソフトウェアの検証を行うようになってきた（例えば、特開平10-149382号）。

【0004】一般に、回路設計を行うために必要となる処理は、図19に示されるように2種類があり、それらは機能検証S101と動作合成S102とである。ステップS101は、そのハードウェア記述（プログラム言語による）をコンパイルして生成した実行形式で機能検証を実行する。ステップS102は、そのハードウェア記述（HDL言語による）を動作合成してRTL（レジスタトランスファレベル）記述を生成するものである。

【0005】ところが、ここで新たな問題が現れた。ソフトウェア用のプログラム言語をそのままハードウェア設計に適用することには不都合がある。例えば、回路の同時並列動作は通常のプログラム言語で記述され得ないという不都合がある。このような不都合を解消してハードウェア設計用のプログラム言語が適正であるように、そのプログラム言語はその動作仕様が拡張され言語拡張されている（発明の実施の形態の項で後述される。）。そのような言語拡張によって、同じハードウェア記述100の動作の解釈が言語拡張をしたプログラム言語で記述したハードウェア記述を用いたS101の処理とHDLで記述したハードウェア記述を用いた動作合成ツールの処理との間で異なっている可能性がある。このような

可能性が含まれているかどうかを確認するために、ステップS101の結果とステップS102の結果との等価性を確認する等価性検証のステップS103が新たに必要である。このような等価性検証により、同じハードウェア記述100の動作の解釈がステップS101のコンパイラとステップS102の動作合成ツールとの間で異なっていることが判明しても、どのようにその解釈の相違が生じたか、どの記述部分でその相違が生じたのかが明らかでなく、人手を用いた多大な時間の投入による解析がハードウェアのデバッグに要する。

【0006】ハードウェア記述のうちプログラム言語とHDLとで解釈が分かれる部分の存在が機能検証シミュレーションを妨害することを回避することが望まれる。特に、そのような等価性検証の不要化が望まれる。

#### 【0007】

【発明が解決しようとする課題】本発明の課題は、ハードウェア記述のうちプログラム言語とHDLとで解釈が分かれる部分の存在が機能検証シミュレーションを妨害することを回避することができるハードウェア記述の検証システム及びその検証方法を提供することにある。本発明の他の課題は、元々の原因がプログラム言語とHDLとの違いを認識できずにハードウェア記述を両言語で記述してしまうことに起因して解釈が分かれるために必要となる等価性検証を不要化することができるハードウェア記述の検証システム及びその検証方法を提供することにある。

#### 【0008】

【課題を解決するための手段】本発明によるハードウェア記述の検証方法は、プログラム言語により記述したハードウェア記述をコンパイルする場合と、HDL言語により記述したハードウェア記述を動作合成する場合とで、論理解釈が異なる部分を検出することである。論理解釈が異なる部分は、ハードウェア記述をシミュレートするコンピュータ設計者によりハードウェア記述のシミュレーションに先だって速やかに論理解釈が正しく確定され得る。

【0009】その解釈が異なる部分は、被代入変数への代入後にその被代入変数を他の数式で参照することが同一のクロックタイミング内で行われる部分である。この場合、プログラム言語により記述したハードウェア記述の複数文の一文を順番に入力し、その一文が被代入変数への代入を行う文であればその被代入変数を信号リストへ登録し、その一文がクロック境界であれば登録されているその被代入変数をその信号リストから削除し、その一文が信号リストに登録されている被代入変数を参照しているかどうかを判断する。その一文が信号リストに登録されている被代入変数を参照している場合に、プログラム言語によるハードウェア記述とHDL言語によるハードウェア記述との解釈が異なる動作となる警告が行われることになる。

【0010】そのような解釈が異なる部分は、同一クロックタイミング内において被代入変数が非オーバーライト型として重ね書きされる部分でもある。この場合、そのハードウェア記述の複数文の一文を順番に入力し、その一文が被代入変数への代入を行う文であれば、その被代入変数を信号リストに登録し、その一文がクロック境界であれば登録されているその被代入変数をその信号リストから削除し、その一文が信号リストに登録されている被代入変数への代入を行っているかどうか判断されることになる。更に、その一文が信号リストに登録されている被代入変数への代入を行っている場合にプログラム言語によるハードウェア記述とHDL言語によるハードウェア記述との解釈が異なる動作となる旨の警告が行われる。

【0011】その解釈が異なる部分は、同一クロックタイミング内でのみ値が有効であり、そのクロックタイミングを越えるとその値が無効になる非代入変数が存在する部分でもあり、非オーバーライト型である被代入変数へ代入された値が同一クロックタイミング内の全ての被代入変数の参照に反映される被代入変数が存在し、且つ、その代入よりもその参照が先にされている部分でもあり、第1オペランドの条件判断が第2オペランドの評価に依存する、両オペランドを用いる演算子が存在する部分でもある。

【0012】本発明によるハードウェア記述の検証システムは、プログラム言語により記述したハードウェア記述の文が一文ずつ入力されるハードウェア記述検証装置と、信号リスト格納装置とを含み、そのハードウェア記述検証装置は、プログラム言語によるハードウェア記述をコンパイルする場合とHDL言語によるハードウェア記述を動作合成する場合との間で動作の解釈が異なる部分を検出し、信号リスト格納装置は、その解釈が異なる部分を格納する。その解釈が異なる部分は、既述の通り例示されている。

#### 【0013】

【発明の実施の形態】図1に一致対応して、拡張プログラム言語により記述したハードウェア記述を検証する検証システムの実施の形態は、ハードウェア記述検証手段・ユニットが信号リスト格納手段・ユニットとともに設けられている。そのハードウェア記述検証手段1は、図1に示されるように、信号リスト格納手段2に双方向に接続している。入力装置3は、記憶装置4に接続している。入力装置3は、プログラム言語により記述されたハードウェア記述5を記憶装置4に出力する。記憶装置4は、入力されたハードウェア記述5を格納して記憶する。

【0014】ハードウェア記述検証手段1は、ハードウェア記述5を記憶装置4から受け取って、後述される部分を検証（検出）したときに、その部分の存在とその存在箇所を信号化した検出信号を警告信号6として出力装

置7に出力する。出力装置7は、その警告信号6を表示する表示装置である。信号リスト格納手段2は、検証対象を信号化した検証対象信号8を格納し、検証対象信号8をハードウェア記述検証手段1に出力する。

【0015】検証対象は、5つの型に分けられる。5つの型は、発明者により、レジスタ型、非オーバーライト型、非レジスタ型、配線型、演算子存在型であるとそれぞれに呼ばれる。

レジスタ型：

10 c l o c k ( )  
x = a + b  
c l o c k ( )  
y = c - d

【0016】このような原プログラムに含まれるxとyは、ともにプログラム上は変数であり、ハードウェアのイメージでは信号であるaとbとにより、被代入変数xが記述されている。被代入変数yは、cとdとにより同様に記述されている。クロックに同期して更新が起こる変数として定義されている被代入変数x、yは、図2と図3にそれぞれに示されるように、クロック11とクロック12とに基づいてレジスタ13とレジスタ14に入力され、異なるクロックタイミングでそれぞれに変化し、その値の更新が代入時ではなくその直後にあるクロック記述に同期して起こる。このようにクロックタイミングで変化する被代入変数のこと（x、y）をプログラム言語上、レジスタ型と拡張的に定義しておく。これによって、実回路上のラッチやレジスタが表現される。

#### 【0017】

レジスタ型検証対象：

30 c l o c k ( ) ;  
x = a + b ;  
y = x + t ;  
c l o c k ( ) ;

この原プログラムに含まれる2つのクロック記述で挟まれる1つの同じクロックタイミング内で代入された被代入変数xがその代入後に他の数式で参照されている。このような形態で2度記述された被代入変数は、HDL記述を用いる動作合成では同じ値ではないと解釈されるが、言語拡張されたプログラム言語を解釈するコンパイラでは誤って同じ値であると解釈される。このような形態で2度以上記述されているレジスタ型被代入変数は、信号リスト格納手段2に格納される。クロックに同期して更新が起こる被代入変数がこのように同一タイミング内で代入された後に他の数式で参照される部分が、レジスタ型検証対象である。

【0018】図5は、レジスタ型検証対象の存在を検出する検出方法を示している。図6は、レジスタ型検証対象を例示している。レジスタ型検証対象である原プログラムの拡張された仕様によるハードウェア記述は、次の通りである。

```

int t;
reg x, y;
x = 0; ... 第1文
clock(); ... 第2文
x = 1; ... 第3文
t = 3; ... 第4文
y = x + t; ... 第5文
clock(); ... 第6文

```

ここで、“reg x, y”と第2文と第6文は、拡張された仕様によるプログラム言語表現である。

【0019】記憶装置4からハードウェア記述の1文ずつがハードウェア記述検証手段1に入力される(ステップS101)。最初の一文は、“x = 0”である。その入力文である第1文が、クロック境界を表す記述であれば、信号リスト格納手段2に格納されている信号情報の全てが削除される。第1文は、クロック境界ではないので、ステップS102からステップS104に進む。ステップS104では、その入力文が信号リスト格納手段2に格納されている信号を参照しているかどうか判断される。今、信号リストには何も登録されていないので、プロセスはステップS106に進む。

【0020】信号リスト格納手段2に格納されている信号を参照している場合は、ソフトウェア実行時に動作が意図しないものになる可能性があるため、その旨を警告としてハードウェア記述検証手段1は出力装置7に出力して、プロセスはステップS106に進む。ステップS106では、レジスタ型信号(x)への代入が存在しているかどうか判断される。第1文にはレジスタ型信号への代入が存在しているので、プロセスはステップS107に進む。

【0021】ステップS107では、レジスタ型信号への代入が存在している場合、その一文で代入されている全てのレジスタ型信号情報を信号リスト格納手段2に格納する。第1文の中のレジスタ信号であるxは信号リスト格納手段2に格納され、プロセスはステップS101に戻る。

【0022】次に、第2文“clock”が、ハードウェア記述検証手段1に入力される。第2文はクロック境界であるから、信号リスト格納手段2に格納されている信号は全てが削除され、ステップS103でxは削除される。プロセスは、ステップS101に戻る。

【0023】次に、第3文“x = 1”がハードウェア記述検証手段1に入力される。第3文は、クロック境界ではないので、プロセスはステップS104に進む。今、信号リストには何も登録されていないので、プロセスはステップS106に進み、ステップS107で信号リストへそのxを登録する。

【0024】次に、第4文“t = 3”がハードウェア記述検証手段1に入力される。第4文はクロック境界ではないので、プロセスはステップS104に進み、第4文

のtはxを参照していないので、プロセスはステップS104からステップS106に進み、ステップS106では、レジスタ型被代入変数への代入は存在しないので、そのtは登録されず、プロセスはステップS101に戻る。

【0025】次に、第5文“y = x + t”がハードウェア記述検証手段1に入力される。第5文は、クロック境界ではないので、プロセスはステップS104に進む。ステップS104で、第5文のyは信号リスト格納手段2に登録されている信号x(第3文のx)を参照しているので、ステップS105で警告が出力され、ステップS106ではレジスタ型被代入変数信号yへの代入が存在しているので、第5文のyはステップS107で信号リスト格納手段2に登録される。次に、第6文がハードウェア記述検証手段1に入力され、xとyとは信号リスト格納手段2から削除される。

【0026】クロック境界内文である第3文と第5文では、第5文の式の被代入変数yが第3文である他の式を参照している。このような場合、C言語等によるコンパイラとHDLでは、図6の表に示されるように、解釈が互いに異なっている。コンピュータ言語として仮に正しくそのプログラムが記載されていたとしても、言語拡張されたC言語によるこの記述と元々のHDLとは、解釈のなされ方が異なってしまうことがある。そのような場合の1つとして、同一のクロックタイミングで代入が行われた被代入変数(同じ文字が用いられている)を他の数式の中で参照する場合がある。

【0027】このような場合、図7に示されるように、第2文のクロックにより、HDLではxには0が代入された状態で第5文を実行するC言語ではxには1が代入されるので、第5文では、C言語ではyは4であるが、HDLではyは3である。このように物理的動作がクロックによりレジスタ等で実行されるクロック境界の前後の文章で、C言語とHDLでは解釈が異なり異なった演算を実行する。このような異なった解釈による演算の実行は、他の式を参照する第5文で結果が異なってしまう。

【0028】

非オーバーライト型検証対象:

```

clock();
z = a + b;
z = c + d;
clock

```

【0029】このような原プログラムは、ハードウェアのイメージでは信号であるaとbとcとdにより2つのz、zが記述されている。このzは、言語拡張において同一タイミング内において、オーバーライトされることがない変数として定義される。これによって、実回路上マルチプレクサを表現する。1つの被代入変数zはaとbとにより記述され、他の1つの被代入変数zはcと

dにより記述されている。クロックに同期して更新が起こる変数として定義されている2つの被代入変数 $z$ 、 $z$ は、コンパイラでは上式の $z$ が下の式の $z$ に代入されて上書きされるので、下の $z$ のみが有効である。動作合成では、図4に示されるように、 $(a+b)$ と $(c+d)$ とがMPX（マルチプレクサ）15により選択されることであると解釈される。非オーバーライト型では、同一クロックタイミング内において同一の被代入変数への代入は2度以上は許されない。

【0030】図8は、非オーバーライト型検証対象の存在を検出する検出方法を示している。図9は、非オーバーライト型検証対象を例示している。非オーバーライト型検証対象である原プログラムの拡張された仕様によるハードウェア記述は、次の通りである。

#### 【0031】

```
ter  z, t;
z = 0; ... 第1文
clock (); ... 第2文
z = 1; ... 第3文
t = 3; ... 第4文
z = t + 2; ... 第5文
clock (); ... 第6文
```

ここで、“ter z, t”と第2文と第6文は、拡張された仕様によるプログラム言語表現である。

【0032】記憶装置4からハードウェア記述の1文ずつがハードウェア記述検証手段1に入力される（ステップS201）。最初の一文は、“z = 0”である。その入力文である第1文が、クロック境界を表す記述であれば、信号リスト格納手段2に格納されている信号情報の全てが削除される（ステップS203）。第1文は、クロック境界ではないので、ステップS202からステップS204に進む。

【0033】ステップS204では、その入力文が信号リスト格納手段2に格納されている信号への代入が存在しているかどうか判断される。今、信号リストには何も登録されていないので、プロセスはステップS206に進む。その代入が存在している場合には、警告が発せられた後にステップS206に進む。

【0034】ステップS206では、非オーバーライト型信号 $z$ への代入が存在しているかどうか判断される。第1文には非オーバーライト型信号 $z$ への代入が存在しているので、プロセスはステップS207に進む。ステップS207では、非オーバーライト型信号への代入が存在している場合、その一文で代入されている全ての非オーバーライト型信号を信号リスト格納手段2に格納する。第1文の中の非オーバーライト型信号である $z$ は信号リスト格納手段2に格納され、プロセスはステップS201に戻る。

【0035】次に、第2文“clock”が、ハードウェア記述検証手段1に入力される。第2文はクロック境

界であるから、信号リスト格納手段2に格納されている信号は全てが削除され、ステップS203で $z$ は削除される。プロセスは、ステップS201に戻る。

【0036】次に、第3文“z = 1”がハードウェア記述検証手段1に入力される。第3文は、クロック境界ではないので、プロセスはステップS204に進む。ステップS204では、今信号リストには何も登録されていないので、ステップS206に進み、ステップS207で $z$ を信号リストへ登録する。

【0037】次に、第4文“t = 3”がハードウェア記述検証手段1に入力される。第4文はクロック境界ではないので、プロセスはステップS204に進む。ステップS204では、信号リストには $z$ のみが登録されているが $t$ は登録されていないので、プロセスはステップS206に進み、ステップS206では、 $t$ はオーバーライト型信号への代入が存在しているので、その $t$ は登録され、プロセスはステップS201に戻る。

【0038】次に、第5文“z = t + 2”がハードウェア記述検証手段1に入力される。第5文は、クロック境界ではないので、プロセスはステップS204に進む。ステップS204で、第5文の $z$ は信号リスト格納手段2に登録されている被代入信号であるので、ステップS205で警告が出力され、ステップS206で非オーバーライト型信号 $z$ への代入が存在しているので、第5文の $z$ はステップS207で信号リスト格納手段2に登録される。次に、第6文がハードウェア記述検証手段1に入力され、 $t$ と $z$ とは信号リスト格納手段2から削除される。

【0039】クロック境界内文である第3文と第5文では、第5文の式の被代入変数 $z$ が第3文の $z$ でオーバーライトされる可能性がある。このような場合、C言語によるコンパイラとHDLでは、図9の表に示されるように、解釈が互いに異なっている。このような場合、第2文のクロックにより、HDLでは $z$ に0が代入されるが、C言語では第3文では $z$ は1、第5文では $z$ は5であるが、図10に示されるように、HDLでは第3文と第5文とのどちらが有効になるか分からないので、このクロック内では $z$ は不明である。このように物理的動作がクロックによりレジスタで実行されるクロック境界の前後の文章で、C言語とHDLでは解釈が異なり異なった演算を実行し、又は、その演算は実行され得ない。このような異なった解釈による演算の実行・不実行は、オーバーライトすると解釈するか、不明として解釈するかで結果が異なってしまう。

【0040】このように、レジスタ型では、信号代入が行われてもすぐには値の更新が行われずクロック境界記述により一斉に値の更新が行われ、非オーバーライト型では、連続したクロック境界記述に挟まれた部分では同一信号に対して複数回の代入を許されない。クロック文章の導入による拡張仕様のプログラム言語では、これに



物理的記述が入り込んでいて、設計者の意図は物理的記述に反映されていない。ハードウェア記述の機能検証と、ハードウェア記述の動作合成の機能検証との等価性の確認をしないで、その等価性が崩れる恐れがある記述部分を検出することにより、原プログラムのその恐れある部分に関して、設計者又はユーザーが解釈に分かれが生じないように書き換える。そのように書き換えられた原プログラムには、解釈の相違は存在しなくなるので、等価性の確認検証は省略され得る。

【0041】以後の型でも以上に既述した同様の各定義を行い、定義にそぐわない各問題を警告する。図11は、非レジスタ型検証対象を例示している。非レジスタ型検証対象である原プログラムの拡張された仕様によるハードウェア記述は、次の通りである。

非レジスタ型検証対象：

```
t = 3 ;
clock () ;
z = t + 2 ;
clock () ;
```

この原プログラムに含まれる  $t$  は、2つのクロックで挟まれる1つの同じクロック境界内で記述されている。3が代入された  $t$  は、そのステップであるそのクロック境界内でのみ有効であり、クロック境界を越えるとその値の3が無効になるタイプの信号であると拡張定義し、非レジスタ型と呼ばれる。

【0042】C言語では以前に代入された値が使用されるが、HDLでは同ステップ内に  $t$  への代入がないので、図12に示されるように、使用される値は合成ツールに依存する。このため、同ステップ内で、且つ、参照以前に代入されていない非レジスタ型信号  $t$  を参照する  $z$  の値は、C言語とHDLでは異なってしまう。図11の表に示されるように、C言語では  $z = 5$  であるが、HDLでは  $z$  は不明である。第3文 " $z = t + 2$ " は、信号リストに登録されていない非レジスタ型信号  $t$  を参照しているので、図13のステップS301とステップS302に示されるように警告が発せられ、非レジスタ型信号への代入が存在しているので（ステップS303）、その非レジスタ型信号  $z$  は信号リストに登録される（ステップS304）。

【0043】図14は、配線型検証対象を例示している。配線型検証対象である原プログラムの拡張された仕様によるハードウェア記述は、次の通りである。

配線型検証対象：

```
t = 3 ;
clock () ;
z = t + 2 ;
t = 1 ;
clock () ;
```

この原プログラムに含まれる  $t$  は、2つのクロックで挟まれる1つの同じクロック境界内で2度記述されてい

る。代入された値がその代入されたのと同クロックタイミング内の全ての参照に反映されるタイプの信号  $t$  は、配線型と呼ばれる。実回路上、信号配線を表現する。

【0044】C言語では以前に代入された値が使用されるが、HDLでは同ステップ内で  $t$  へ代入された値が使用される。図15に示されるように、 $t$  に代入される値は、以前に代入された値の3であるか、新たに代入される値の1であるかが、C言語とHDLとは異なってしまう。図14の表に示されるように、C言語では  $z = 5$ 、 $t = 1$  であるが、HDLでは  $z = 3$ 、 $t = 1$  である。第3文 " $z = t + 2$ " は、信号リストに登録されていない配線型信号  $t$  を参照しているので、図16のステップS401とステップS402に示されるように警告が発せられ、配線型信号への代入が存在しているので（ステップS403）、その配線型信号  $z$  は信号リストに登録される（ステップS404）。

【0045】図17は、演算子型検証対象を例示している。演算子型検証対象である原プログラムの拡張された仕様によるハードウェア記述は、次の通りである。

演算子型検証対象：

```
i = 0 ;
a = 0 ;
clock 0 ;
if (i > 0 && a++) {
    i = 0 ;
}
clock 0 ;
```

【0046】まず、 $a++$  とは、これを評価すると  $a$  をインクリメントすることを意味する。この原プログラムに含まれる演算子  $\&\&$  は、C言語では演算子  $\&\&$  の左オペランド ( $i > 0$ ) の条件が真である場合にのみその右オペランド  $a++$  を評価するが、HDLでは演算子  $\&\&$  の左オペランドの条件の真偽に係わらず左右のオペランドを並列に評価する。演算子  $\&\&$  の左オペランドの条件が偽の場合に、右オペランドを評価するかどうかの点が、C言語とHDLとは異なる。図18に示されるように、演算子  $\&\&$  が使用されていて（ステップS501）、演算子  $\&\&$  の右オペランド中に変数値の更新を伴う記述が存在する場合（ステップS502）、警告が出される（ステップS503）。演算子  $\&\&$  に代えられて左オペランドの条件が偽の場合にのみ右オペランドを評価する演算子が用いられている場合にも、図18に示されるように警告が発せられる。

【0047】

【発明の効果】本発明によるハードウェア記述の検証システム及びその検証方法は、ハードウェア記述のうちプログラム言語とHDLとで解釈が分かれる部分を検出することにより、そのように解釈が分かれる部分の存在が機能検証シミュレーションを妨害することを警告するこ

とによってプログラム修正を容易化し、ひいては回避することができる。特に、等価性検証を省略することができる。

#### 【図面の簡単な説明】

【図 1】図 1 は、本発明によるハードウェア記述の検証システムの実施の形態を示すシステムブロック図である。

【図 2】図 2 は、ハードウェア記述図である。

【図 3】図 3 は、他のハードウェア記述図である。

【図 4】図 4 は、更に他のハードウェア記述図である。 10

【図 5】図 5 は、本発明によるハードウェア記述の検証方法の実施の形態を示すフロー図である。

【図 6】図 6 は、両言語の記述の内容を示すデータ表である。

【図 7】図 7 は、更に他のハードウェア記述図である。

【図 8】図 8 は、本発明によるハードウェア記述の検証方法の実施の他の形態を示すフロー図である。

【図 9】図 9 は、両言語の記述の他の内容を示すデータ表である。

【図 10】図 10 は、更に他のハードウェア記述図である。 20

【図 11】図 11 は、両言語の記述の内容を示すデータ表である。

【図 12】図 12 は、更に他のハードウェア記述図であ

る。

【図 13】図 13 は、本発明によるハードウェア記述の検証方法の実施の更に他の形態を示すフロー図である。

【図 14】図 14 は、両言語の記述の内容を示すデータ表である。

【図 15】図 15 は、更に他のハードウェア記述図である。

【図 16】図 16 は、本発明によるハードウェア記述の検証方法の実施の更に他の形態を示すフロー図である。

【図 17】図 17 は、両言語の記述の内容を示すデータ表である。

【図 18】図 18 は、本発明によるハードウェア記述の検証方法の実施の更に他の形態を示すフロー図である。

【図 19】図 19 は、公知のハードウェア記述の検証システムを示す回路システム図である。

#### 【符号の説明】

1…ハードウェア記述検証手段

2…信号リスト格納手段

3…入力装置

4…記憶装置

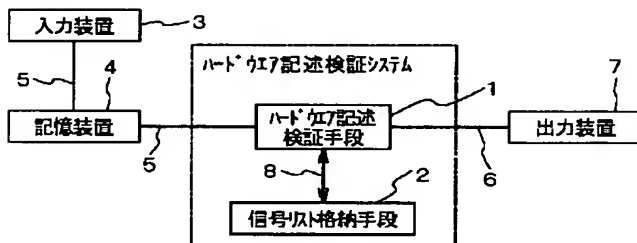
5…ハードウェア記述

6…警告信号

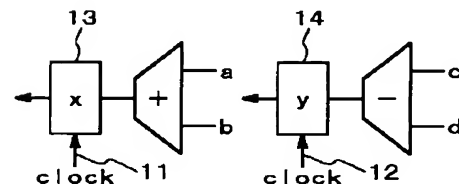
7…出力装置

8…検証対象信号

【図 1】

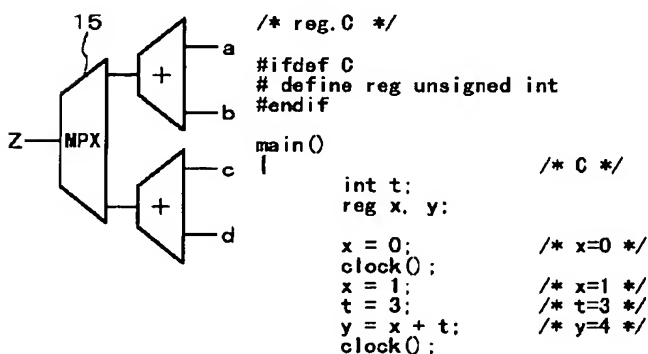


【図 2】



【図 3】

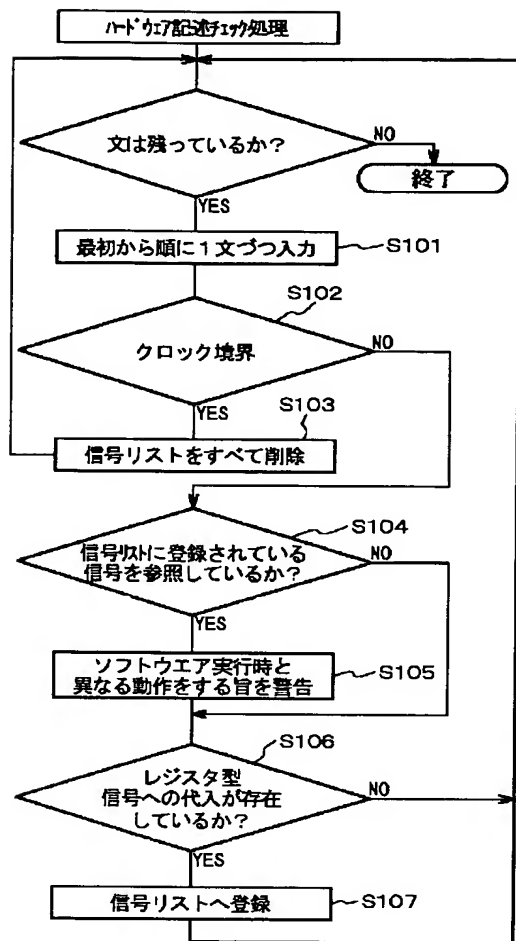
【図 4】



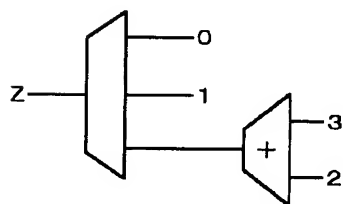
【図 6】

/* HDL */	/* LIST */	
/* x=0 */	/* {x} */	/* S107 */
/* t=3 */	/* {t} */	/* S103 */
/* x=1, Y=3 */	/* {x, y} */	/* S105, S107 */
	/* {t} */	/* S103 */

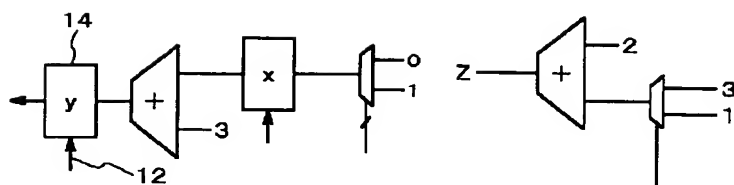
【図 5】



【図 10】

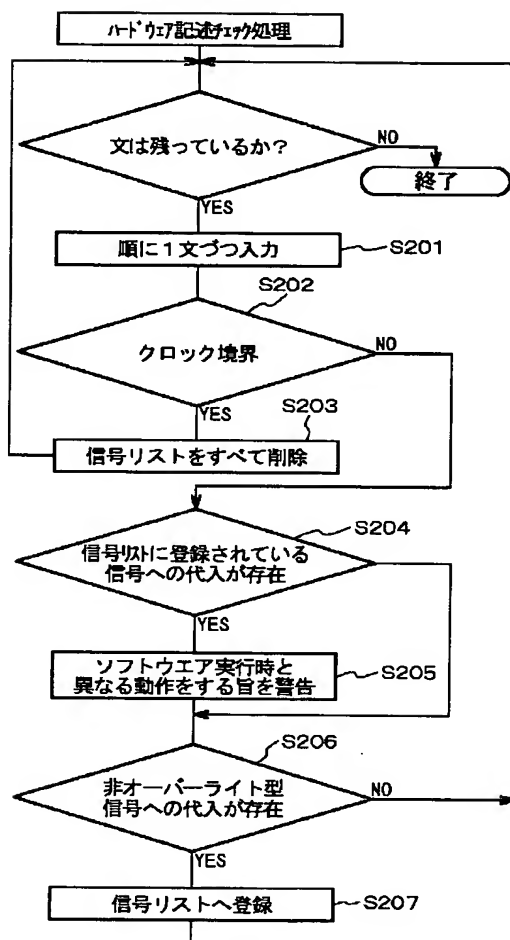


【図 7】



【図 15】

【図 8】



【図 11】

```
/* ter.C */
```

```
#ifdef C
# define ter unsigned int
#endif
```

```
main()
```

```
ter z, t;
```

```
t = 3;
```

```
clock();
```

```
z = t + 2;
```

```
clock();
```

```
/* C */
```

```
/* t=3 */
```

```
/* z=5 */
```

```
/* HDL */
```

```
/* t=3 */
```

```
/* t=? */
```

```
/* z=? */
```

```
/* LIST */
```

```
/* {t} */
```

```
/* {z} */
```

```
/* { } */
```

【図 9】

```

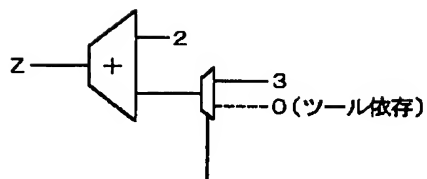
/* assign.C */
#ifdef C
#define ter unsigned int
#endif

main()
{
    ter z, t;          /* C */          /* HDL */          /* LIST */

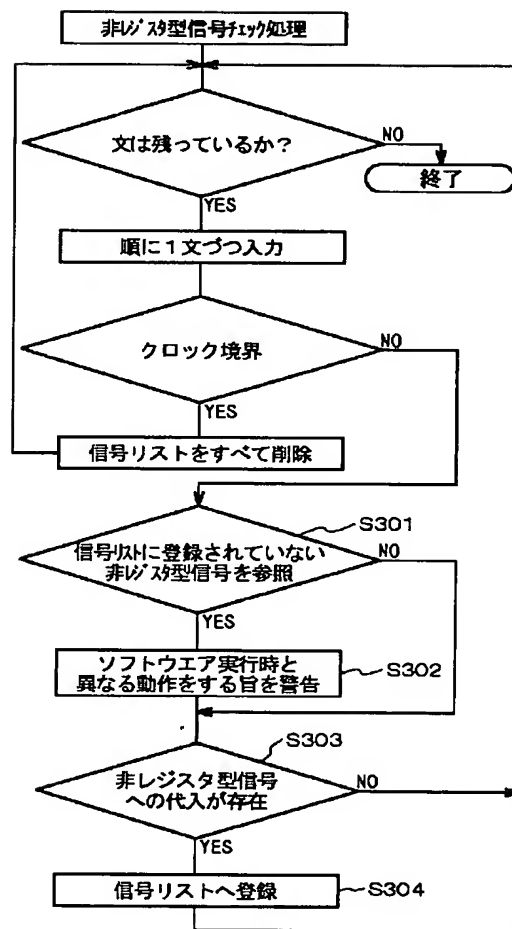
    z = 0;              /* z=0 */          /* z=0 */          /* {z} */          /* S207 */
    clock();            /* z=1 */          /* z=? */          /* {} */          /* S203 */
    t = 3;              /* t=3 */          /* t=3 */          /* {z, t} */       /* S207 */
    z = x + t;          /* z=4 */          /* z=? */          /* {z, t} */       /* S205, S207 */
    clock();            /* S203 */
}

```

【図 12】



【図 13】



【図14】

```

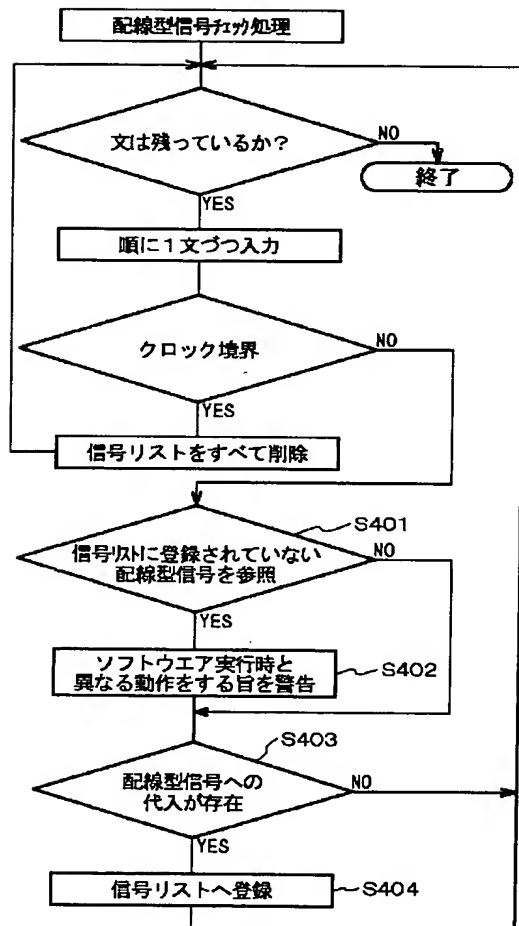
/* wire.C */
#ifdef C
#define ter unsigned int
#endif

main()
{
    ter z, t;          /* C */      /* HDL */      /* LIST */

    t = 3;              /* t=3 */      /* t=3 */
    clock();            /* z=5 */      /* z=3 */      /* {t} */
    z = t + 2;          /* t=1 */      /* t=1 */      /* {} */
    t = 1;              /* t=1 */      /* t=1 */      /* {z} */
    clock();            /* t=1 */      /* t=1 */      /* {z, t} */
}

```

【図16】



【図17】

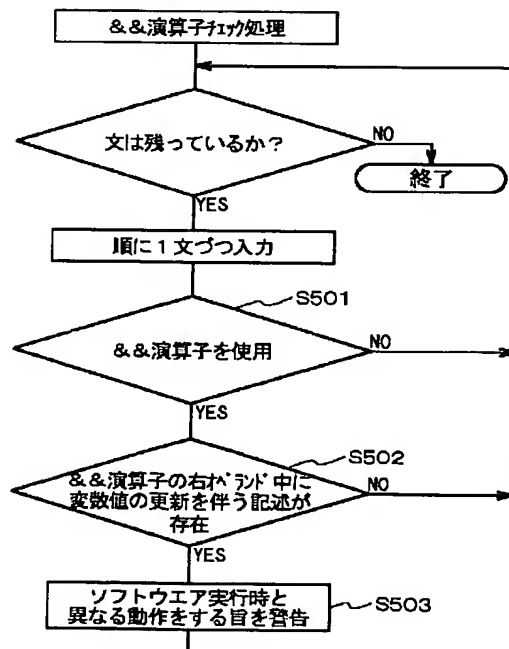
```

/* and.C */
main()
{
    int a, i;          /* C */      /* HDL */

    i = 0;              /* i=0 */      /* i=0 */
    a = 0;              /* a=0 */      /* a=0 */
    clock();            /* a=0 */      /* a=1 */
    if(i > 0 && a++) {  /* a=0 */      /* a=1 */
        i = 0;
    }
    clock();
}

```

【図18】



【図 19】

